Problem Identification & Solution Proposal

1. Problem Identification: Data Inconsistencies and Missing Fields

Issue:

The media content stored in Elasticsearch contains unstructured or missing fields, which can lead to inconsistent search results and a poor user experience. For example:

- Most documents has incomplete metadata (e.g., missing titles, descriptions).
- The DB field values are not always be "st" or "sp"(the value is mostly “stock” as per I checked it), leading to incorrect URL generation.
- Media IDs are not always be 10 characters long, causing issues with URL padding.

Impact:

Data inconsistencies can result in:

- Inaccurate or incomplete search results.
- Broken image URLs, leading to a degraded user experience.
- Difficulty in maintaining data quality over time as more media items are added.

2. **Proposed Solution: Data Normalization and Validation**

Solution Overview:

To address data inconsistencies, we need to implement a robust data normalization and validation process when retrieving and displaying media content.

Data Validation: Ensure that all required fields are present and correctly formatted.

Data Transformation: Normalize data to improve searchability and consistency.

Error Handling: handle missing or invalid data.

I have adjusted the model in the backend to accommodate the current structure of the data:

```
2 usages  manju
public MediaItem(String id, String db, String text, String photoGraphers, OffsetDateTime datum) {
    this.id = id;
    this.db = db;
    this.text = text != null ? text : "";
    this.photoGraphers = photoGraphers != null ? photoGraphers : "";
    this.thumbnailUrl = buildUrl(this.db, this.id);
    this.datum = datum;
}

1 usage  manju
public String buildUrl(String db, String id) {
    String paddedId = String.format("%010d", Integer.parseInt(id));
    return "https://www.imago-images.de/bild/" + (db != null ? db : "st") + "/" + paddedId + "/s.jpg";
}
```

Justification

This solution is effective because it ensures that the system can handle unstructured or missing data gracefully, improving both the reliability and user experience of the application. By validating and normalizing data, we can:

- Prevent broken image URLs and display consistent search results.
- Maintain data quality over time as new media items are added.
- Simplify debugging and maintenance by logging errors and providing fallback values.

Scalability & Maintainability

Scalability Considerations:

1. Indexing Strategy: As the volume of media content grows, consider implementing an efficient indexing strategy in Elasticsearch to ensure fast search performance. Use appropriate mapping types and optimize the index settings for large datasets.
2. Caching: Implement caching mechanisms (e.g., Redis) to store frequently accessed media records and reduce the load on Elasticsearch(for now I have already implemented Spring @Cacheable in the backend to handle this).
3. Load Balancing: Use load balancers to distribute incoming requests across multiple backend servers, ensuring high availability and scalability.

Maintainability Considerations:

1. Modular Code Structure: Organize the code into modular components (e.g., data validation, transformation, error handling) to make it easier to maintain and extend.
2. Documentation: Provide clear documentation for the codebase, including comments, README files, and API documentation.
3. Version Control: Use version control systems (e.g., Git) to manage changes and collaborate with other developers.

Monitoring & Testing

Monitoring:

1. Logging: Implement logging throughout the application to capture important events and errors. Use a centralized logging system (e.g., ELK Stack) to aggregate and analyze logs.
2. Performance Metrics: Monitor key performance metrics (e.g., response times, error rates) using tools like Prometheus and Grafana.

Testing:

1. Unit Tests: Write unit tests to verify the correctness of individual components (e.g., data validation, transformation functions).
2. Integration Tests: Perform integration tests to ensure that different components work together as expected.

3. End-to-End Tests: Conduct end-to-end tests to simulate user interactions and validate the overall functionality of the system.